

Title	コンピュータ・ネットワークのソフトウェア
Author(s)	山本, 欣子
Citation	大阪大学大型計算機センターニュース. 17 p.15- p.30
Issue Date	1975-05
oaire:version	VoR
URL	<a href="https://hdl.handle.net/11094/65270">https://hdl.handle.net/11094/65270</a>
rights	
Note	

*Osaka University Knowledge Archive : OUKA*

<https://ir.library.osaka-u.ac.jp/>

Osaka University

## コンピュータ・ネットワークのソフトウェア

財団法人  
日本情報処理開発センター 山 本 欣 子

### 1. 序 論

コンピュータ・ネットワークの言葉の定義は現在まだ確立されていないが、ここでは複数の独立したコンピュータ間で、ハードウェア、ソフトウェア、データ・ベースなどを共用するリソース・シェアリング (Resource sharing) コンピュータ・ネットワークを対象とし、図1に示すような分散型異機種結合のネットワークを考える。

この様なネットワークは次の3つの要素から成り立つ。

- (1) HOST コンピュータ (図のH) ; それぞれ異ったリソースを持つ独立したコンピュータ・システム
- (2) NODE (図のN) ; HOST あるいは端末群 (図のT) をネットワークに結合する節となるところで、一般にこれ自身がコンピュータ機能を持つ。
- (3) 回線 ; NODE間を結ぶ通信回線

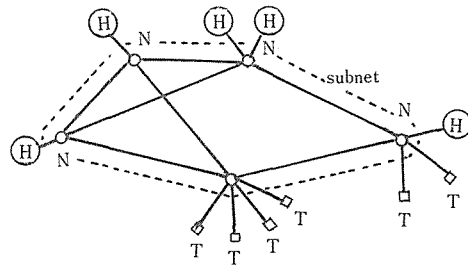


図1 分散型ネットワーク

図1に示すように NODEと回線を総称してサブネット (Subnet) と呼ぶ。

このサブネットはネットワーク内のすべてのデータ転送をつかさどる交換網であり、回線交換、パケット交換、メッセージ交換等、いろいろと現在議論のあるところだが、ここでは、この種のネットワークで現在広く利用されている store & forward 機能を持つパケット交換方式を対象とする。

このようなネットワークで、NODEにコンピュータ機能を持たせる理由は

- ① 異機種 HOST間のインターフェースの調整のしやすさ
- ② HOST 負荷の軽減
- ③ ネットワーク拡張の容易さ
- ④ ネットワークの信頼性の向上

等である。例えば、HOST同士を直接結合するとすれば、各 HOSTは相手の HOSTの1つ1つに対するインタフェースを考慮せねばならず、HOSTの負荷が増大する。どの HOSTもN-

ODE プロセッサに対するインタフェースのみを考えればよいという事は新しい HOST を結合するのも容易であり、ネットワークの拡張性の面からも望ましい。これはまた経済性の問題でもある。ARPANET の経験からも言われるようにネットワーク全体の不安定性は、その多くが HOST の不安定性から来るという。したがって、ネットワークに参画しているいくつかの HOST の障害時に於て、ネットワーク自体への影響をできるだけ少なくするにはデータの転送機能を HOST からはなるべく切り離れた方が全体の信頼性が向上するという事が言える。

さて次に、ネットワークでは避けて通れない話、即ちプロトコルというものについて少しふれておこう。

いくつかの異った要素からなるシステムに於ては、各要素の役割と分担を明確にするための規約をもうける必要があり、これをプロトコル (protocol) と呼ぶ。図 1 に示すようなネットワークに於けるプロトコルは通常次の 5 つのレベルの階層構造をなしている。

(1) 隣接 NODE 間プロトコル

メッセージ・フォーマット、伝送手順、確認、障害監視等に関する規約。

(2) 発信地 NODE — 目的地 NODE プロトコル

発信地と目的地 NODE 間のメッセージ転送の確認、再送の手順等に関する規約。

(3) NODE — HOST プロトコル

サブネットと HOST の間のデータ授受に関する規約。

(4) HOST — HOST プロトコル

HOST 同士が交信し、相互の処理の依頼およびデータ転送をおこなう手順に関する規約。

(5) 高位プロトコル

ネットワーク・ユーザが直接使用するコマンドを規定したもので、TSP (TSS protocol), RJEP (Remote Job Entry protocol), FTP (File Transfer Protocol) 等が基本的なものである。

このうち(1)~(4)がシステム・レベルのプロトコルで(5)はユーザ・レベルのプロトコルである。これらは(1)から(5)へ順次階層構造になっており、この事はあるレベルのプロトコルを変更しても他のレベルのプロトコルには影響を与えないという意味もある。

コンピュータ・ネットワーク・システムに於けるこれらのプロトコルの設定はきわめて重要な問題であり、プロトコルの設定、即ちネットワークの機能設計と考えても良い。したがってネットワークの論理機能というものは、プロトコルという形でとらえる方がより厳密であると思うが、このプロトコルをハードウェアとの協力によりコンピュータ上にフィジカルにインプリメントするのがソフトウェアの役割であり、ここではプロトコルという耳なれぬ言葉を避けて、一応ソフトウェアという言葉を使っているが、むしろネットワーク・コントロールのための論理的機能の話という様に受けとっていただければ有難い。

## 2. ネットワークのコントロール機能

ネットワークのコントロール機能は、サブネットの各 NODE プロセッサと各 HOST とで分担しておこない、ソフトウェアとしては NODE プロセッサ内の SCP (Subnet Control Program) と各 HOST 内の NCP (Network Control Program) とに分かれる。両者の負荷分担はネットワーク設計上の大きな問題の 1 つであり、これについては、夫々具体的な話の中でも触れるが、前述のようにデータ転送の信頼性および効率にかかわる一切の処理は SCP の分担とし、逆に各 HOST に依存する部分は SCP からは除外し、各 HOST に吸収する。また、異機種ネットワークに於てもすべての NODE の SCP は同一のものとするのが拡張性の点からも望ましいと言われている。

## 3. SCP

サブネットのコントロールに対するハードウェアとソフトウェアとの分担がまず 1 つの問題であり、これについては現在きわめて流動的である。

ネットワークシステムの普及により、NODE プロセッサのハードウェア機能が現在きびしく再検討されており、従来の汎用小型機に、ROM、マイクロプログラム等の機能追加による体質改善や、専用のマイクロプロセッサの利用、あるいは従来のソフトの割り込み処理をハード化したものとも言われる Lockheed Sue の出現など今後の通信制御用ハードウェアは大きく変化してゆくであろう。

一方 VAN (Value Added Network) や特殊通信網に見られるように、更に一步進んで網に組み込まれた交換機能としての NODE は、もはや独立したコンピュータという形態さえ失い始めている。

このような現状をふまえて、ここでは一応 SCP をソフトウェアとして話を進めるが、今後これらの機能の一部はハード化が可能であろう。

さて、SCP 設計上いくつか留意すべき点がある。

- ① 信頼性の面から、NODE と HOST とは互いに依存関係がなく、また NODE 相互間にも依存関係がないこと。即ち、障害の発生した HOST や NODE はそれを切り離した形でネットワーク自体の運転は継続せねばならない。
- ② 効率の面からは会話型処理の応答速度は充分早く、且つ大量データ転送に於ては充分なスループットが得られること。

ネットワークを通して会話型処理を行っている端末ユーザが、例えば 2 秒以内のレスポンスを要求した場合、2 秒のうち 1 秒がその会話型のサービスをおこなっている HOST の周りのローカルな部分 (即ち local network) で消費されたとすると、サブネット内は少なくとも 1 秒以内で通過せねばならない。例えば ARPANET ではサブネット内の消費時間は 0.2 秒程度におさえている。

このように、1 つのメッセージに着目した場合の伝送時間を問題にする場合と、大量のデ

ータ転送では全体としてのスループット、即ち各 NODE プロセッサで単位時間当りにどれだけの伝送処理が行えるかを問題にする場合との両面があるが、両者を満足させる1つの手段として、使用目的に応じてメッセージの長さにより長短のバラエティを持たせ、夫々にふさわしいコントロール方法を採用するという事が一般におこなわれている。

- ③ 拡張性の面からは、ネットワークの拡張やトポロジの変更に対し、影響を受けぬソフトウェアであること。

以下に SCP の代表的な機能について述べる。

#### (1) Store & Forward (蓄積交換)

NODE 間プロトコルに基づく隣接 NODE 間および発信地、目的地 NODE 間のメッセージの伝送手順、その確認、再送手続き等、ネットワークの基本となるメッセージの伝送処理である。

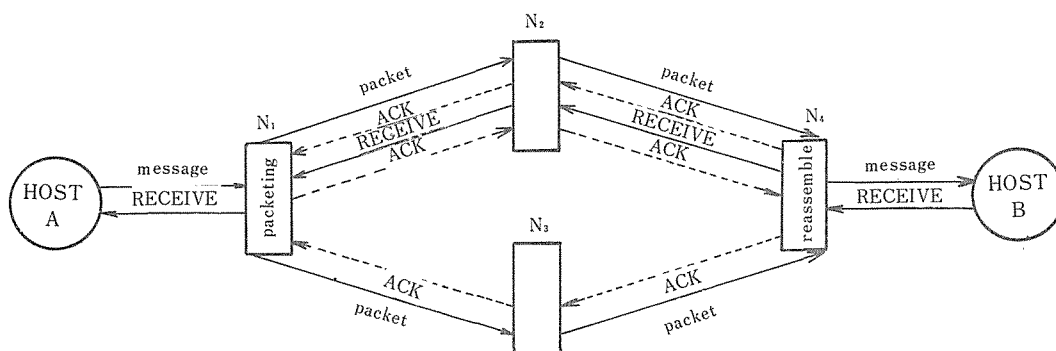


図2 蓄積交換

図2に示すように、HOST-AからHOST-Bに送られるメッセージは発信地NODE  $N_1$  でパケット (packet) に分解され、各パケット毎に任意の経路を経て、目的地NODE  $N_4$  に伝送され、ここでリアセンブル (reassemble) され、再びメッセージとなってHOST-Bに送られる。ここでいうメッセージとはサブネット上を伝送されるコントロール情報およびデータのすべてを含む。

さて、このメッセージとパケットの関係であるが、メッセージというのは一般に任意の長さを持っており、伝送の際にはこれを適当な大きさに分割した方が処理しやすく、効率の面からも望ましい。この分割したものがパケットである。このパケット化とリアセンブルをサブネットで行うか、HOSTで行うかは現在議論の分かれるところである。もしこれをHOSTで行えば、サブネットの負担は後で述べるフロー・コントロールなどとも関連し、非常に楽になる。ARPANETはサブネットで、フランスのCycladesはHOSTでこれを行っている。パケット受信の確認として隣接NODE間ではACKが返され、発信地と目的地間ではRECEIVEが返される。これもRECEIVE返送をNODE間で行う場合とHOST間で行う場合とがある。図一2はHOST間で行っている例であり、ARPANETではNODE間でRFNM (Request For Next

Message) を返している。

ACK や RECEIVE などの返送により、そのためのトラフィック・オーバーヘッドが増加するのはある程度止むを得ないが、できるだけこれを防ぐため、数個分まとめたり、逆方向のパケットに便乗させる等の手段が一般にとられる。

なお、これらのメッセージのフォーマットや伝送手順は国際的に標準化される傾向にあり、現在 IBM などから手順に関する提案が出されている。またこれらの蓄積交換機能は、ハード化の可能性が比較的強い部分といえよう。

## (2) Routing (ルーティング)

発信地 NODE から目的地 NODE へメッセージを伝送するルートを決めることで、次のような要求がある。

- ① 最短時間で目的地に達すること。
- ② ネットワークの拡張やトポロジの変化に影響せぬアルゴリズムであること。これは実際の拡張や変更だけではなくネットワークの 1 部、例えば、ある NODE やある回線の障害により一時的にネットワーク・トポロジが変化することを含む。
- ③ トラフィックの変動によく追従しうること。
- ④ NODE の負担が軽いこと。これにはアルゴリズムの複雑さによる処理負荷と、パケット・コピーの保存のためのスペース上の負荷とがある。
- ⑤ ルーティング情報伝達のためのトラフィック・オーバーヘッドを過度に増加させぬこと。
- ⑥ ルートを平均的に使用すること。

分散型ネットワークでは発信地 NODE から目的地 NODE へ到着する経路は最低二通りはあるというのが特徴である。これが集中型より信頼性が高いといわれる所以でもあるのだが、実はそのためにどこを通過してゆけば一番良いかというルートを選択する仕事が必要となる。ルーティングの研究というのは、コンピュータ・ネットワーク以前からも種々行われてきたが、現在ネットワークで採用されている代表的な手法は固定法 (Fixed routing) と適応法 (Adaptive routing) である。

### A. 固定法 (Fixed routing)

目的地までのホップ数 (通過する NODE の数)、距離、各 NODE における予測しうるディレイ (delay) の性質等から予め各目的地への経路を定めてしまう方法である。予測し得るディレイとして、例えば大型 HOST やアクセスの頻度の高いデータ・ベースにつながる NODE 附近をなるべく避けて、すべての回線をできるだけ平均的に使うという様な配慮が行われることもある。

予めルートが固定してしまっているため、処理のオーバーヘッドは殆んどないが、トポロジの変化、トラフィックの一時的な変動等には適応性がない。

### B. 適応法 (Adaptive routing)

トラフィック量の変化、一部の障害によるネットワーク・トポロジの変化などを反映させ、

ダイナミックにパケットのルートを適応させてゆく方法である。前述のように、分散型ネットワークでは原則として NODE は少なくとも二つの出力ラインを持つ。すべての他の NODE に対してどの出力ラインを通せば、それぞれどれだけのディレイがあるかというディレイ・テーブル (Delay table) を各 NODE が備えており、その時点で最小のディレイを示す出力ラインを選ぶ。このディレイ・テーブルの更新のメカニズム、あるいはダイナミックなルーティング情報の入手方法等によりいくつかの種類がある。

1. Isolated local estimate: すべてのパケットに発信地を出た時から現在に到る経過時間を記録させることにより、逆に各ノードでその発信地までの時間をその記録から推定する。

“Backward learning”とも呼ばれる。

2. Shortest queue + Bias (SQ + B): その時点における NODE 内部の各出力ラインに対するそれぞれのキュー (Queue) の長さと、すべての他の NODE への到着時間の表 (一般にはスタティックな情報に基づく)、即ちバイアス・テーブル (Bias table) との兼ね合いでルートを決定する。

3. Distributed updating: 各 NODE はすべての NODE へゆくその時点での最短時間を記録した表 (Update vector) を持っており、それを隣の NODE へ適当な時間間隔で送り、隣の NODE はそれに自分の NODE の条件による補正、即ち自 NODE 内の各ラインへのキューの状態や、隣の NODE と自分の間のディレイ等により補正を加え、また更に次の NODE へと順次最新ルーティング情報を波及させてゆく。

これらの適応法はいずれも固定法にくらべ状態の変化に適合してゆく利点があるが、処理負荷およびトラフィックのオーバ・ヘッドはある程度避けられない。例えば、ARPANET は Distributed updating 法を使用しているが、ルーティング情報は全トラフィックの 2 ~ 3 % を占めるといふ記録もある。

以上のようなルーティング手法にはまた多くのバリエーションがみられ、またどのようなルーティング手法を採用するかはネットワークの規模やトラフィックの特性によるが、Fultz のシミュレーション<sup>(1)</sup>によると SQ + B が比較的良好なパフォーマンスを得ている。しかし、一方固定法の方が適応法よりも優れているという意見もあり、また同じネットワークであってもトポロジを変化させることにより、各ルーティング手法のパフォーマンスの差も大きく変化してしまうという実験もあるようである。

なお、適応法ではメッセージが同一個所を繰り返し回ってしまうルーピング (looping) や、2 つ NODE 間を行ったりきたりするピンポン現象を起こす恐れがあり、その対策を考える必要がある。例えば、各パケットは一度通過した NODE には再び渡さない、あるいは少なくとも隣から受けたパケットは再びそれへ戻すことは禁止する、または各パケットにホップカウンタ (hop counter) を設け、それが全 NODE 数 +  $\alpha$  以上になっているパケットはルーピングの可能性があると見て、それをみつけた NODE は棄却する等の方法がある。

(3) Flow control (フロー・コントロール)

サブネットへのメッセージの流入を無制限に許すとサブネット内のコンジェスション (congestion) や各種のロック・アップ (lock up) が発生し、ネットワークが硬直状態に陥いる可能性がある。これを回避するためサブネットへ流入するトラフィックを規制するのがフロー・コントロールである。

フロー・コントロールには2つの代表的な考え方がある。1つは目的地にそのメッセージを受け入れる態勢が整っているという確認を得て初めてメッセージの伝送を開始するという考え方である<sup>9)</sup>。具体的には目的地 NODE におけるメッセージ受け入れ用のバッファ・エリアの予約をとるという事となる。これは主として複数パケットからなる長いメッセージのリアセンブル用の要求である事が多い。例えば、リアセンブル・ロック・アップとはマルチ・パケット・メッセージの伝送に於て目的地 NODE で1つのメッセージの全パケットが揃わぬうちにバッファが一杯となり、いつまでもリアセンブルが完了しないためバッファが空かずロック・アップが生ずる状態をいう。これはそのメッセージのリアセンブル用に予め必要なバッファを予約しておけば防ぐことができる。

もう1つの考え方は、サブネットに流入するパケットの総量を何等かの形で規制するという基本的な考え方であり、いろいろな形で実現される。例えば、NPL の Davies の提案による Isarithmic (ギリシャ語で equal number の意) という考えでは<sup>12)</sup>、まずあるメッセージがサブネットに入る時は必ずパーミット (permit) と呼ぶ通行許可証を得なければならない。発信地 NODE でパーミットを得たメッセージは目的地 NODE でそのパーミットを返す。サブネット内のパーミットを一定にしておけば、総量が規制できるというものである。

また別の方法としては、すべての発信地 NODE、目的地 NODE 間にパイプ (pipe) と呼ぶロジカルなパスを考え、そのパイプの容量を制限することによって1度にサブネットに流入するパケット量を規制するという考え方もある。

サブネット内の流れが滞る原因は概して NODE 内のバッファ・エリアの余裕とトラフィックの集中の度合によるものであり、バッファ・コントロールのアルゴリズムがフロー・コントロールに大きな影響を与える。例えば、Store & Forward ロック・アップは2つの中継 NODE 間で両方向のトラフィックが集中している場合、両 NODE とも既にバッファが1杯になっていると互いに相手からのパケットを受け入れる事ができず、同時に既に送ったパケットの ACK も受けとれない。ACK が返れば保管していたそのコピーが捨てられるのでフリー・バッファ が得られるが、それにもかかわらず両者立ちすくみの状態になる場合である。このロック・アップへの対策としては最低1個のバッファを常に逆方向用にリザーブしておくだけで充分効果がある。

いずれにしろ、フロー・コントロールは全体としてのトラフィックをある程度規制した上で、更にバッファ管理の行き届いたメカニズムによる立体的な両面作戦が必要であろう。

#### (4) シークエンシング (Sequencing)

これは伝送されたメッセージを正しい順序で目的地 HOST に送り込む機能である。もし、前のメッセージの到着確認の RECEIVE を受け取ってから、始めて次のメッセージを発信すると



いうプロトコルのネットワークであれば、このシーケンシングは必要ない。しかし同一目的地に複数個のメッセージを一時に発信し得る場合はサブネット内のトラフィックの状態、伝送エラー等により、先に送り出したメッセージが必ずしも先に目的地に到着するとは限らない。この場合、一般には目的地 NODE で到着したメッセージのシーケンシングを行い、HOST には正しい順序で送り込む。

通常、メッセージにはメッセージ番号がつけられ、これにしたがってシーケンシングが行われる。例えば、1, 2, 3, と番号のついた3つのメッセージのうち、1も2も到着しないうちに3がまず到着しても、目的地 NODE は一時それを保留して、1, 2 の到着を待ち、HOST には、1, 2, 3 の順で送り込まねばならない。この場合は少なくとも3つのメッセージを入れる待ち合わせバッファ・エリアが必要となる。

フロー・コントロールの手段でもあるパイプの概念は、実はシーケンシングにも大きな機能を果たす。パイプの容量を例えば4とすると、ある一対の NODE 間で一度に伝送し得るメッセージ数は4個までであり、従ってシーケンシングのためのバッファは最大4メッセージ分確保しておけばよい。

#### (5) 障害対策

サブネットの障害対策に対する基本的態度は次のようなものである。

- ① フェイルソフト (fail soft) 機能が万全であること。即ち前述のように障害部分を取り除いた環境下で可能な限り運転が継続されること。
- ② ネットワークを構成する夫々異った要素間、例えば HOST とサブネット、NODE と回線等の障害時の切りわけが容易に行われること。
- ③ 障害発生やその回復手続きの過程に於て、他の部分の運転に影響を与えぬこと。
- ④ 障害の検出、その回復のテスト等が可能な限り自動的に行われること。

障害対策の主な対象は次のようなものである。

1. メッセージの紛失： 伝送の確認は隣接 NODE 間では ACK が返され、発信地、目的地間では RECEIVE が返される。タイマーによって一定時間経過しても、これらの確認の返送がない場合は手持ちのコピーを再送したり、メッセージ番号を指定して問い合わせが行われる。
2. メッセージの重複： シーケンシングの際にメッセージ番号により発見され、重複したものは棄却される。
3. 回線および NODE の障害： 回線の障害は一般にハードウェア割り込みで検出される。また NODE の障害は隣接 NODE により発見される。例えば20~30パケットあるいは一定時間の連続伝送に対し、何の応答もない場合に障害と見なすという様な方法がとられる。NODE や回線の障害は他の NODE に通知され、障害 NODE 宛のパケットは途中で棄却される。
4. HOST 障害： NODE から送ったメッセージを一定時間経過しても HOST が受けとらぬ場合、即ち Write コマンドが一定時間内に終了しない場合、HOST 障害とみなす。障害 HOST 宛のパケットは棄却される。

5. ディスコネクションの検出： ネットワーク内で、いずれのルートをとっても相互に通信不能の NODE が生じた場合、これをディスコネクション (Disconnection) 状態という。また、互いに孤立した NODE 群同士もディスコネクション状態である。図3にそれらの例を示す。

(a)では6が、(b)では1, 2, 3

と4, 5, 6が互いにディスコネクション状態である。

サブネット内に発生したディスコネクションを放置すると、孤立した NODE, または NODE 群に向けられたパケットは行き先がないため、サブネット内を無駄に回り続け有効なトラフィックの妨げ

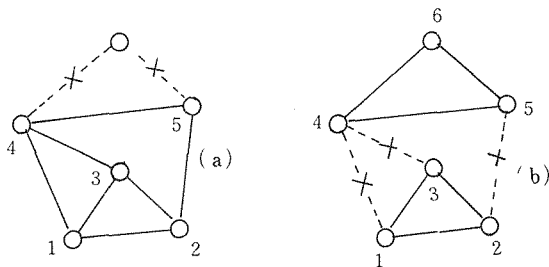


図3 ディスコネクションの例

となる。そこでディスコネクションの状態は出来るだけ早期に発見することが望ましい。

発見の方法としては基本的には2つの考え方があり、ネットワークのトポロジを各 NODE に与える方法と、与えぬ方法とである。後者の例として ARPANET で採用している方法は、ルーティング情報の周期更新を行う際に各 NODE は自分を通してすべての目的地へ行く最小ホップ数をのせて周辺 NODE に知らせる。それを受けた各 NODE は更にそれに1を加えて次の NODE に送る。ディスコネクション NODE が発生すると、その目的地に対するホップ数が際限なく大きくなり、これが (全 NODE 数 - 1) を越えたらその NODE はディスコネクション状態と判断する。

また、前者の例としては、図4に示すように各 NODE は直接ラインが張られている NODE に対する要素を1、張られていない要素を0としたネットワーク・トポ

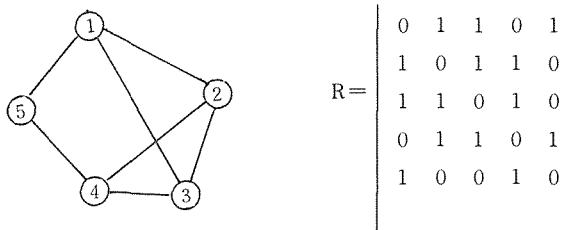


図4 コネクション・マトリックス

ックス R を持ち、音信不通になった NODE をその隣接 NODE がとらえてこの R を更新し、それを次の NODE にディストリビュートする。 $R^n$  の各要素  $rij$  は  $i$  から  $j$  に  $n$  ホップで行く道があるかないかを表わすというグラフ理論の原理を利用し、各 NODE は更新された R をもとに全 NODE 数を  $n$  とした場合、 $R^{n-1}$  の計算を行い、自分の NODE から見て何処がディスコネクションであるかを検出する<sup>[6]</sup>

以上 SCP の代表的ないくつかの機能につき述べたが、この他ネットワークの稼動状況のモニタリング機能、各種障害の記録を行う管理機能、サブネット・テストの為の擬似トラフィック発生機能、NODE 障害時あるいは SCP 更新時に有効な隣接 NODE からの SCP プログラムのセルフローディング機能等、各種の付加機能が考えられる。

#### 4. N C P

NCPは各 HOSTに存在するネットワークのコントロール・プログラムである。一般に NCPは各 HOSTの OSの一部、あるいはそれに準ずる形で存在することとなる。

異機種結合によるネットワークでは、それぞれまったく独自の思想によって作られている異った OSとのインタ・フェースを十分考慮したうえで、NCPをどのように実現すべきかを検討せねばならない。

NCPの機能は前述の HOST—HOST プロトコルや、HOST—NODE プロトコルの規定に沿ったものである。その主な機能を以下に述べる。

##### (1) HOST間のコミュニケーション

HOST間のコミュニケーション機能は、そのネットワークのサービス機能の如何によっていささか異なるが、ここでは一応バッチ処理、リモート・バッチ処理、TSS処理のすべてを含む汎用の場合を考えよう。

各 HOSTのジョブの処理単位は、ジョブ、ジョブ・ステップ、タスク等と一般にそれぞれ異なる可能性があるが、ネットワーク上ではいずれもこれをプロセス (process) という概念で統一する。そして HOST間コミュニケーションは実際にはプロセス間コミュニケーションという形に置き換えて実現する。すなわち、プロセス間コミュニケーションは

- ① 各 HOSTのバッチ、リモート・バッチ、TSS等の各処理単位を一元的に扱うことを可能とするとともに
- ② 他 HOSTのプロセスも自 HOSTのプロセスに準じた形で扱い得る機能を持つ。

プロセス間コミュニケーションには、以下の3つの機能が含まれる。

##### ① プロセス間のコネクションの確立と閉鎖

図5に示すように、2つの HOST-A、Bのそれぞれのプロセス a、bがおのおの NCPを通して相互に通信するには、まず両者の間でコネクションを確立せねばならない。コネクションの確立とは、一方の入力が他方の出力となるロジカルなパスを作ることである。各 HOSTに於て夫々のプロセスにつける id

の規則はまちまちであり、それをネットワーク

では一元化し、グローバルな idをつける。これを port-idといい、コネクションは2つの port-id、ここでは X、Yの pair に対して確立される。ARPANETでは port-idを socket 番号と呼ぶ。このロジカルなパスを閉じるのがコネクションの閉鎖である。

##### ② プロセス間のデータの授受

コネクションが確立すると、それを通して2つのプロセス間でデータの授受がおこなわれる。

##### ③ プロセス間の同期

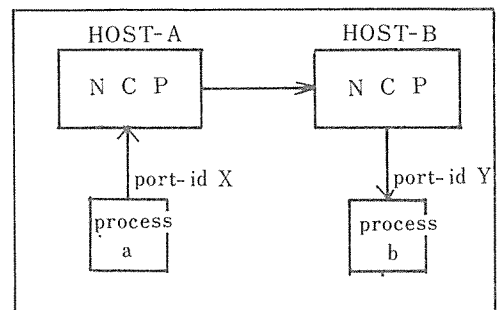
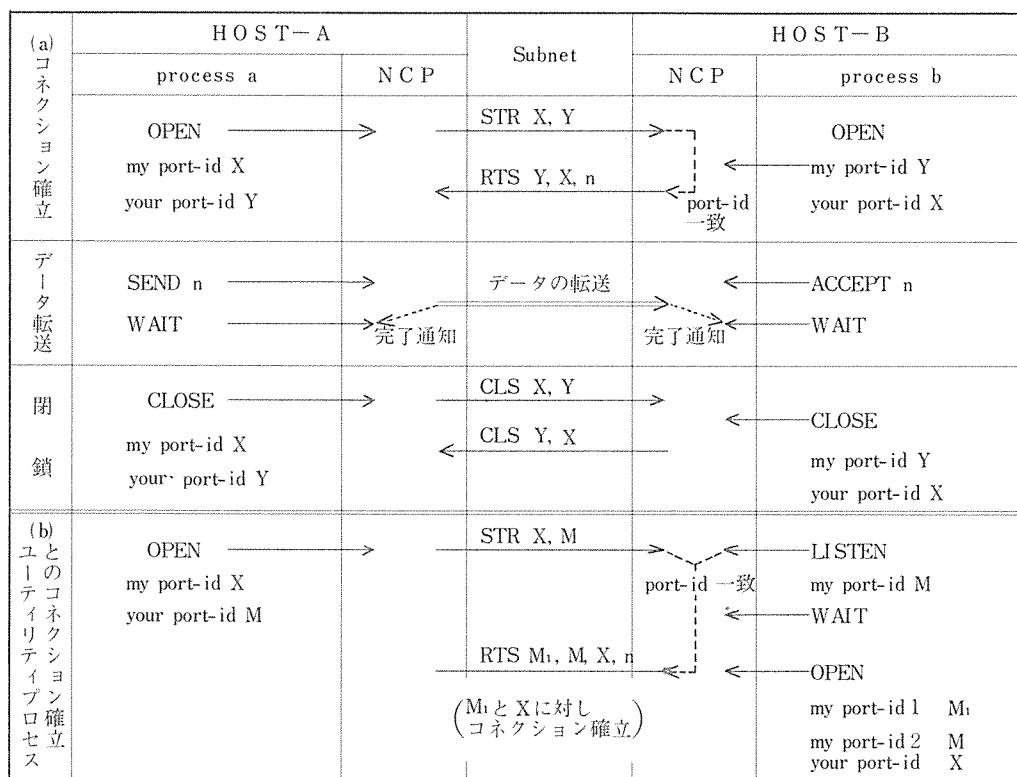


図5 プロセス間コネクション

コントロール コマンド	
STR	コネクション確立要求
RTS	コネクション確立応答
CLS	コネクションの閉鎖
CEASE	データ送信の一時停止
RESUME	データ送信の再開
POST	コントロール情報の通知
LOAD	プロセスの起動
DEL	プロセスの消滅

サービス コマンド	
OPEN	コネクション確立要求
GLOSE	コネクションの閉鎖
ACCEPT	コネクションを通してデータを受ける
SEND	コネクションを通してデータを送る
LISTEN	コネクション確立要求待ち
WAIT	イベント待ち
POST	コントロール情報の通知

図6 コントロールコマンドとサービスコマンドの例



(n はコネクション番号)

図7 両コマンドの関連と使用例

ネットワークを通して複数の HOST で 1 つの仕事をおこなう Cooperated job processing に於てはプロセス間の同期をとる必要がある。

単発の短いメッセージ転送を主体とするネットワークでは、予めコネクションを確立するという手続きを省き、データ送受信の具体的要求が出た時点で、その都度コネクション確立に相当する手続きをとる方法もある。しかし大量データの連続転送や、頻度の高いデータ転送で

は、かえってオーバ・ヘッドが増える可能性もある。また1つのコネクションに1方向のデータ流のみを許す場合と両方向のデータ流を許す場合とがある。ARPANETは1方向であるがCycladesは両方向であり、後者を推す人も多い。

#### (2) コントロール・コマンドとサービス・コマンド

2つのHOSTのNCP間はコントロール・コマンド、即ちHOST-HOSTプロトコル・コマンドにより交信される。一方、プロセスとNCP間にはユーザのためのサービス・コマンドを設ける場合があり、サービス・コマンドには多重コネクション管理、HOST間のフロー・コントロール等のマクロな機能を含ませる。図6に両コマンドの種類を、図7に両者の関連と使用法の例を示す。(b)はMがユーティリティ・プロセスの場合である。一般にユーティリティ・プロセスはMの他に複数個のport-id, 例えばM<sub>1</sub>, M<sub>2</sub>, M<sub>3</sub>...を所有し、この例ではM<sub>1</sub>とXのペアに対してコネクションが確立されている。

#### (3) HOST間のフロー・コントロール

サブネットのフロー・コントロールとは別に、HOST間にもフロー・コントロールが必要である。各HOSTは一般に複数HOSTと同時に交信しており、1つのコネクションにおいて送信側HOSTが大量のデータを送り込んでも、受信側は常にそれにマッチした速度で処理し得るとは限らない。

HOST間のフロー・コントロールは基本的にはNCP内のバッファの利用状況の管理の問題となる。

バッファ管理の一例としてバッファ・プール(buffer pool)の考え方がある。各NCPはそれぞれ適当な個数のバッファを持つ。HOST間で1つのコネクションが確立すると、最低1個の基底バッファが与えられる。この基底バッファはそのコネクションがクローズするまで保持される。それ以上は必要に応じてバッファ・プールから取り出して供給され、使用後はプールに返却する。プールに残ったバッファ数が、ある限度を切った場合、多量にバッファを占有しているコネクションに対しては先方のHOSTにデータ送出の一時停止を要求する。このコネクションに対してはHOST内の処理が進行し、その基底バッファの内容が処理のために明け渡された時点で送信が再開される。

いずれにしろ、コネクションが少ない時は多くのバッファが利用でき、混雑時にも少なくとも1個の基底バッファは確保できるような配慮が望ましい。バッファの個数は一般に1つのHOSTに於て一時に確立し得るコネクション数(例えば24とか32)と、平均メッセージ長、バッチ処理とTSS処理の比率等に基づき割り出される。

#### (4) 高位プロトコルの処理

高位プロトコルは、ネットワーク・ユーザが直接使用されるマクロなコマンドとしてサポートされる。例えば、前述のTSSプロトコル、RJEプロトコル、ファイル転送プロトコル等はいっしょに基本的なものである。ネットワーク・サービスの拡大により、これらの高位プロトコルは数も種類も増加してゆくものであり、その処理形態も拡張性を十分考慮に入れる必要がある。

る。

各 HOST 内プロセスには、ユーザ独自のユーザ・プロセスとシステムがサポートするユーティリティ・プロセスとがあり、例えば高位プロトコルの処理を高位プロトコル処理プロセスというユーティリティ・プロセスの形でおこなうのは 1 つの方法である。

図 8 に TSP (TSS protocol) 処理の概念図を示す。HOST-A の TSS 端末から(図 8 の左下) HOST-B の TSS 処理を使用するには、HOST-A の NCP のもとに TSP プロセスというユーティリティ・プロセス(2 重の枠)があり、それに対し HOST-B の NCP のもとにも仮想端末プロセスが存在する。HOST-A, B とも既存の OS の機能として TSS モニタがあり、例えば HOST-B における通常の会話型使用は HOST-B の下の端末(点線)から B の TSS モニタを経由してユーザ・ジョブと会話を行うわけであるが、ネットワークを通して HOST-A の端末から HOST-B のこのユーザ・ジョブと会話をする場合は点線で示した通常の端末へのつながりが、B の TSS モニタの中で仮想端末プロセスに切り換えられ、両 HOST の NCP を通して会話が行われることとなる。

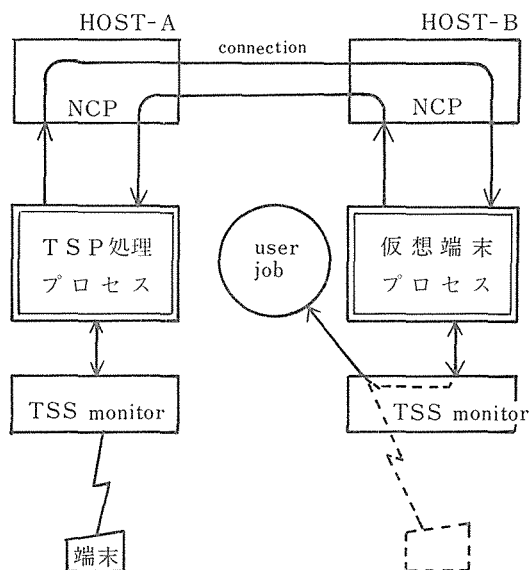


図 8 TSP プロセス処理概念図

図 9 は RJEP (Remote Job Entry Protocol) プロセス処理の概念図であり、夫々の HOST に USER RJEP プロセスと SERVER RJEP プロセスが存在する。この図の①～⑰の番号は処理のコントロールの順序を示す。

図 10 および 11 は対応する TSS の初期会話と RJE ジョブデックの例である。今後の傾向としてはこれらの図の □ 内の HOST オリエンテッドな部分が NCL (Network Control Language) または NAL (Network Access Language) というような形で統一化される傾向にある。とくに最近ネットワーク同士の結合、あるいは国際ネットワーク等の話題もクローズアップされつつあるが、その様な動向も含めてのネットワーク利用の汎用言語の出現が期待される。

#### (5) NCP 作成

Subnet の SCP と HOST の NCP を比べてみるとサブネットの SCP は将来 VAN や特殊網に吸収され得る可能性もあるが、NCP の作成は今後とも残される問題となるであろう。特に既存の OS とのインタ・フェースを十分検討し、NCP の肥大化による HOST の過度の負担を

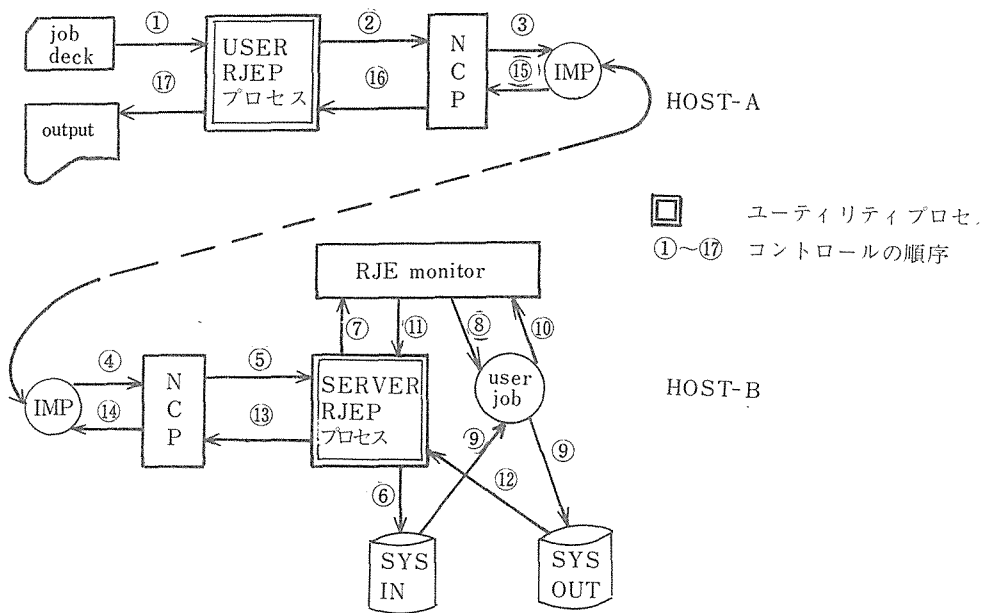


図9 RJEプロセス処理概念図

```

START Keyを押す
HELLO
TSP
NETWORK USER No. =××××
NETWORK PASSWORD =
HOST No. =B
CONNECT TO HOST B
HOST-Bとの会話
... (下線はユーザの入力)

```

図10 端末からの会話

```

¥ID ××××
# USER No. ××××
# HOST B
# OUTPUT S
# END
HOST-BのJCL
¥JEND

```

←HOST-AのJCL

} RJE コマンド

←HOST-AのJCL

(OUTPUT Sは結果を Source(HOST-A)に返す事を指示)

図11 RJEのジョブデック

防ぎ、かつオーバーヘッドの増加を最低限におさえねばならぬ。

既存 OS に対する NCP の位置づけは、次の3通りが考えられる。

- ① 制御プログラムの一部とする。
- ② ユーザ・プログラムとする。
- ③ ①、②の中間的なサブ・モニタの形をとる。

これらのうち①はある意味では理想的であるかもしれぬが、ネットワーク結合のため増加した制御プログラム機能が、その HOST のみによるローカル・ジョブのオーバーヘッドの増加をも

たらず可能性がある。また、既存 OS 部分に改造を加えねばならぬ可能性も強いが、いわゆる標準 OS ではなくなるため、その後のメーカーのメンテナンス・サポートが期待できぬ恐れがある。

これに対し②のユーザ・プログラムとして作成する形はユーザとしてはもっとも作成が容易であるが、ユーティリティ・プロセスの起動や終了などのコントロール能力を同じレベルのプログラム同士では発揮できぬ可能性があり、また割込みやタイマの処理、制御テーブル内の情報の参照など、かなりの無理が生ずる可能性が強い。

以上のような点を総合すると、結果的には③のサブ・モニタの形で NCP を作成するのが最も望ましいこととなるが、いずれにしろこれらの選択は既存の OS の機能および性格によって決定付けられてしまい選択の余地がない可能性も強い。NPL の Davies 等は、将来の OS は NCP の下に従来の OS が従属すべきであると提唱しているが<sup>(4)</sup>、少なくとも今後の OS は他のコンピュータとのコミュニケーションを前提として設計すべきであるということであろう。たとえば、他のコンピュータとのコミュニケーションを可能とするには、最低限以下のような機能が既存 OS にそなわっていることが望ましい。

① 多重処理機能をそなえていること。

② 結合用アダプタがペリフェラルの一種として扱い得ること。

ビット・ストリング・データのトランスペアレント (transparent) な受け入れが可能であ

③ ること。

④ NCP とユーザ・プロセスの間で情報の交信が可能なこと。

⑤ NCP がプロセス (ジョブ) の起動、監視および強制終了を行えること。

⑥ プロセスに必要なファイルが動的に割り付けられ、また消去も可能であること。

⑦ 言語プロセッサ、ユーティリティ・プログラム等、既存の処理プログラムが NCP から利用できること。

⑧ アカウンティングなどのためにリソース利用情報が NCP から参照できること。

## 参考文献

- (1) Fultz, G. L. Adaptive routing technique for message switching computer communication networks UCLA-ENG-7252, July 1972.
- (2) Davies, D. W. The control of congestion in packet-switching networks. Trans. IEEE, Vol. COM-20, No.3, June 1972.
- (3) Kahn, R. E. and Crowther, W. R. Flow control in a resource sharing computer network. Proc. Second ACM IEEE Symposium on problems in the optimization of data communication system, Palo Alto, Calif. Oct. 1971.
- (4) Davies W. D. and Barber D. L. A, Communication networks for computers. John Wiley & Sons 1973.



- (5) Puzin, L. Presentation and major design aspects of the Cyclades computer network. 3rd. Data Comm. Symp. Tampa, Nov. 1973.
- (6) (財)日本情報処理開発センター, コンピュータ・ネットワーク JIPNETの研究開発, 49-S 001, 1975年3月。